N74-72607

A LATTICE THEORETIC SOLUTION TO A PROBLEM IN MEMORY RE-ALLOCATION[†]

by

Mark S. Tuttle

9-72

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts 02138

[†]This research has been supported in part by the National Aeronautics and Space Administration under Grant NGR-22-007-176, and by the Division of Engineering and Applied Physics, Harvard University.
These results were announced at the Princeton Conference on Information Sciences and Systems, March, 1971.

Abstract

A memory re-allocation problem posed by (Knuth) and (Even, Lempel, and Pnueli) is solved using lattice-theoretic techniques. The problem involves finding the minimum cost relocation of stacks in memory which will satisfy new memory requirements. The number of stacks currently residing in memory and the order in which they are stored are assumed to be fixed. The solution requires "time" which is between linear and quadratic in the number of stacks. In "well-behaved" multi-stack (or multi-program) environments, the "time" to find a solution will be nearly linear in the number of stacks.

The main result is generalized to show how lattices may be extracted from permutation-graphs.

Contents

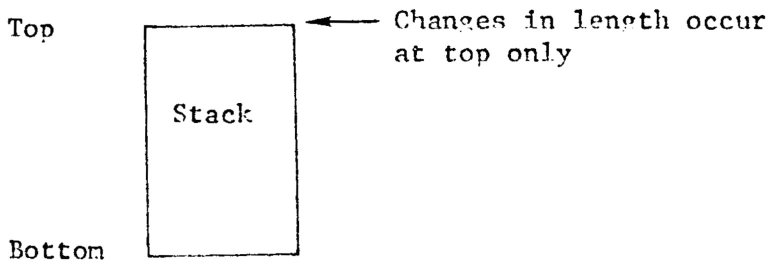
0.0 Re-allocating Memory*

If the number and space requirements of various lists, tables, pages, programs, etc. stored in a computer's addressable memory vary with time and contiguous memory locations are used to store the contents of each list, table, page, program or whatever; so that the lists, tables, pages, and programs can be viewed as simple "objects" of varying length distributed throughout memory then some scheme is needed to re-allocate space whenever objects compete for the same space or when new objects need to be stored in memory. Specifically, it may be necessary to move some of the objects "up" or "down" in memory, or "out" of addressable memory to make room for each other's growth, or to make room for new objects.

*A constrained version of this problem was given to A. Pnueli at the Weizmann Institute, Rehovet, Israel by some designers of a large multi-program computer. (Shimon Even, Personal conversation, April, 1971.)

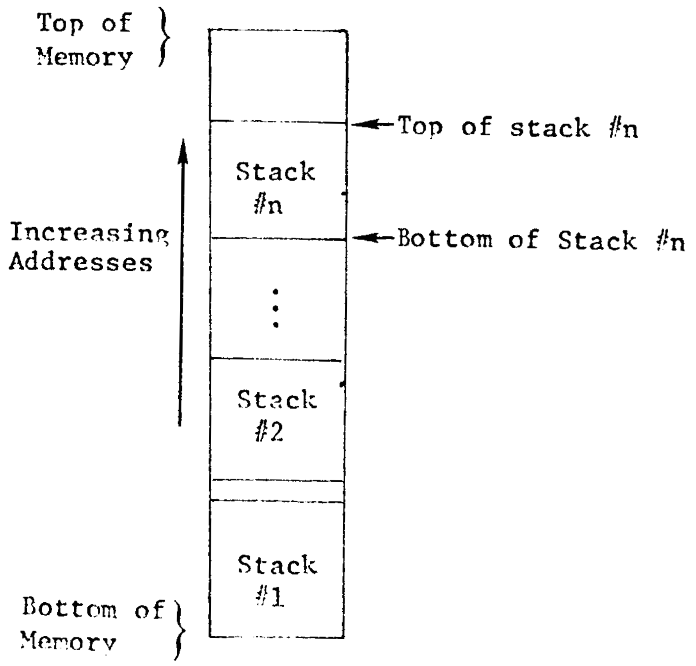
0.1 The Model

All changes in length of any memory object occur at one, and only one, end of that object so that each object can be modeled by a stack (K).



All stacks in memory are oriented in the same direction, so that the address of the bottom of a stack is always less than or equal to the address of the top of the stack (K).

The current content of memory consists of n stacks, labeled with the integers 1,2,...,n starting from the bottom of memory (E,L,&P).



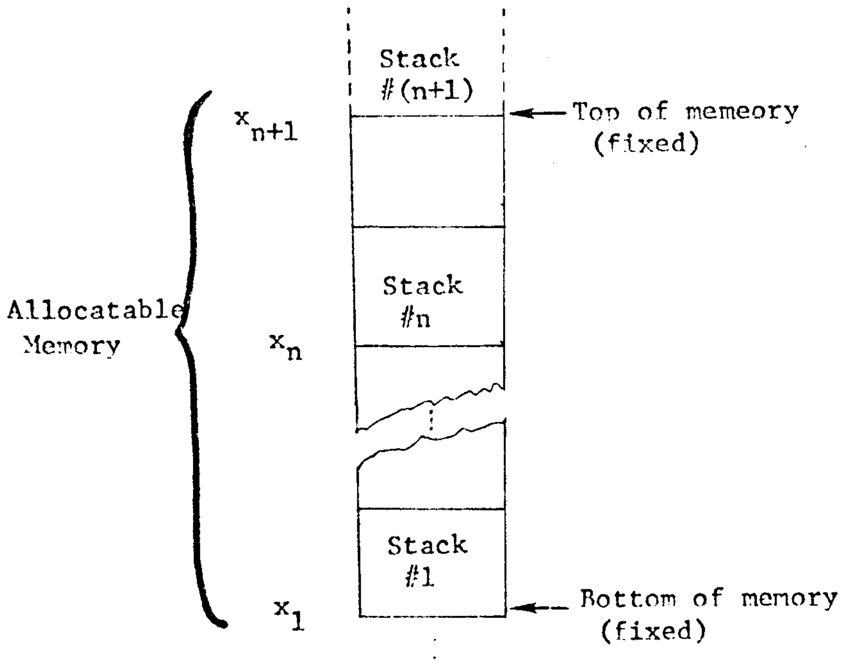
Addressable Memory

The allocatable portion of memory is bounded by two addresses.*

x_1 ≡ constant; the bottom of memory; the address of the first available location in memory

x_{n+1} ≡ constant; the top of memory; the address of the first location which is not allocatable

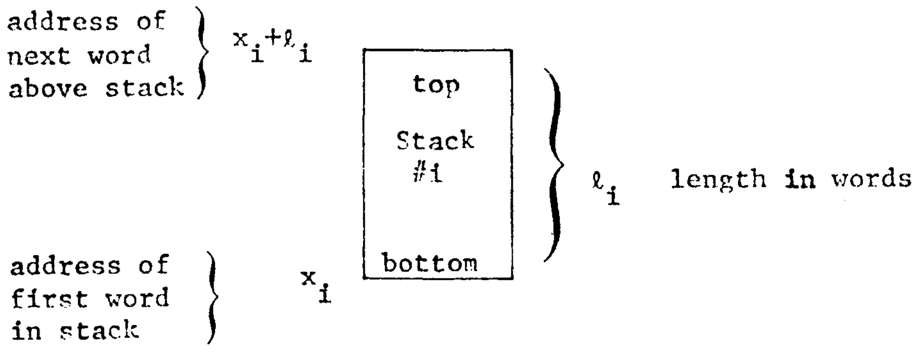
Thus stack #1 is not relocatable but can have varying length. Stack #(n+1) is not relocatable and is "dummy" in the sense that its length is irrelevant to the re-allocation problem.



*These boundary conditions, which differ slightly from those used by (K), or those suggested by (E,L,&P), were chosen because they are more consistent with the model being formulated.

The current memory configuration is described by the following sets.

$N_r = \{2,3,\ldots,n\}$	is the set of relocatable stacks, where the integer $i \in N_r$ is the label of the ith stack from the bottom of memory.
$X_r = \{x_2,x_3,\ldots,x_n\}$	is the set of starting addresses of the relocatable stacks, where $x_i \in X_r$ is the starting address of stack $i \in N_r$.
$L = \{\ell_1,\ell_2,\ldots,\ell_n\}$	is the set of lengths of the variable length stacks, where $\ell_i \in L$ is the non-negative current length of stack $i \in N_r \cup \{1\}$.


address of next word above stack
$x_i+\ell_i$
top
Stack #i
bottom
ℓ_i length in words
address of first word in stack
x_i

Note that the model allows stacks of length zero. Thus if $\ell_i = 0$, then it is possible that $x_i = x_{i+1}$; in other words, empty stacks do not occupy space.

After a certain amount of time the memory requirements of the n variable length stacks currently residing in memory change, and it may be necessary to relocate some of the stacks to accomodate the new memory requirements. The new memory requirements of the n stacks are specified by the set $L' = \{\ell_1', \ell_2', \ldots, \ell_n'\}$, where $\ell_i' \in L'$ is the new length (memory requirement) of stack $i \in N_r \cup \{1\}$ (E,L,&P).*

0.2 A Cost Function

w is any non-negative function which assigns to each relocatable stack in addressable memory a cost of relocation which is independent of the distance moved.** Thus $w(i)$ is the cost of moving stack $i \in N_r$ to some other part of addressable memory. If a re-allocation of memory requires that a set of stacks $S_r \subseteq N_r$ be relocated (and thus that the set $(N_r - S_r) \cup \{1,n+1\}$ remain in place), then the cost of re-allocation is $\sum_{i \in S_r} w(i)$ (E,L,&P). Occasionally the notation $w(S)$ will be used where $w(S)$ is defined to be $\sum_{i \in S} w(i)$.

*L and L' distinguish the current from new (or "successor") memory requirements. In all other cases, the prime symbol will indicate the modification of a set to include reference to elements 1 and n+1. Example: If $N \subseteq N_r$ is a set of stacks, $N' = N \cup \{1,n+1\}$.

**This constraint on the cost function reflects the behavior of "move" and "block transfer" instructions on most contemporary computers. The time required to move a stack is linear with length (of the stack) on almost all machines.

0.3 Relocation Without Re-ordering

The definition of the re-allocation problem adopted by (K) and (E,L,&P) imposes the following constraints:

1) the number of stacks in addressable memory remains fixed;

2) the order that the stacks are stored in addressable memory remains fixed.

Thus the memory re-allocation problem reduces to a problem of relocation without re-ordering.

1.0 Finding a Minimum Cost Re-allocation

1.1 The Optimization Problem

Given the model specified in 0.1, the memory relocation problem can be formulated as a discrete optimization problem. The problem is to choose a subset S_r of the set of relocatable stacks N_r which minimizes the sum $\sum_{i \in S_r} w(i)$ and which does not violate the constraints (1)-(4) below.

1) Constant Number--n, the number of variable length stacks, remains fixed (K).

2) Constant Order--the order in which the stacks are stored in memory remains fixed (K).

3) Capacity--It is always true that

$$\left[\sum_{i=1}^{n} \ell_i'\right] \leq (x_{n+1}-x_1).$$

(Otherwise, the new memory requirements exceed the capacity of memory (F,L,&P).)

4) No Overlap--Let $S_s' = (N_r - S_r) \cup \{1,n+1\}$ be the set of stacks which remain stationary. Then it must always be true that if $i,j \in S_s'$ and $i < j$, then

$$\left[\sum_{k=i}^{j-1} \ell_k'\right] \leq (x_j - x_i).$$

The "no overlap" constraint asserts that between any two stacks i and j which are not relocated there is enough room for the new memory requirements of stacks $i, i+1, \ldots, j-1$ (F,L,&P).

Alternatively, a subset $S_s \subseteq N_r$, a subset of relocatable stacks which are to remain stationary, can be chosen which maximizes the sum $\sum_{i \in S_s} w(i)$ and which does not violate the constraints (1)-(4), where $S'_s = S_s \cup \{1,n+1\}$ is used to define S'_s in constraint (4).

Claim: The problem of finding a set of stacks $S_s \subseteq N_r$ which can remain stationary, and which has maximum relocation cost, is the dual of the problem of finding a set of stacks $S_r \subseteq N_r$ to be relocated which has minimum relocation cost.

Proof. $N_r = S_r \cup S_s$ and N_r is fixed.

Definition: Given sets X_r and L', the ordered pair $M = (X_r, L')$ is a memory relocation problem.

Definition: Given a memory relocation problem M, a solution (i.e., a solution to M) is a maximal* set of stacks $S_s \subseteq N_r$ which can remain stationary.

Definition: Given a memory relocation problem M and a relocation cost function w, a relocation optimization problem is an ordered pair (w,M).

*Given a set S and a predicate P, S is maximal under P if and only if $P(S)$ and $\exists$ no set T such that $S \subset T$ and $P(T)$, where $\subset$ denotes proper containment.

Definition: Given an optimization problem (w,M), a solution $S_s \subseteq N_r$ to the relocation problem M, has maximum relocation cost if and only if for all solutions $T_s \subseteq N_r$ to M,

$$\left[\sum_{i\varepsilon S_s} w(i)\right] \geq \left[\sum_{i\varepsilon T_s} w(i)\right].$$

Definition: Given an optimization problem (w,M), a max-cost solution is a solution $S_s \subseteq N_r$ which has maximum relocation cost.

Proposition 1: Given only a cost function w --for all subsets of stacks $S_s \subseteq N_r$, there exists a memory relocation problem M, such that S_s is a max-cost solution to the relocation optimization problem (w,M).

Proof. Choose* the sets X_r and L' so that S_s is the only solution to $M = (X_r,L')$, and then S_s is necessarily the max-cost solution to (w,M).

Proposition 1A: Given "condition X" (see below), a cost function w, and a set of starting addresses X_r; for each subset of stacks $S_s \subseteq N_r$, there exists a set of new memory requirements L' such that S_s is a max-cost solution to the relocation optimization problem (w,M), where $M = (X_r,L')$.

*Assuming $(x_{n+1}-x_1) > n$, where n is the number of variable length stacks.

Condition X: For most sets X_r likely to arise in practice "condition X" can be eliminated from Proposition 1A. It could probably be eliminated entirely if the memory reallocation problem were generalized to allow stacks to disappear when empty (i.e., generalized to let n be variable), since most of the anomalies occur when several stacks are empty or when the number of stacks approaches the number of storage locations in memory. In any case, condition X indicates that the current problem definition is not entirely free or redundancy (since it is reasonable to expect that the set X_r is available long before the set L', and for some sets X_r not every solution $S_s \subseteq N_r$ is possible).

The version of condition X below is sufficient, but certainly not necessary, for Proposition 1A to be true. The proof given below is, accordingly, less satisfying than one which employed a necessary, as well as sufficient, condition X.

"Condition X" is true if and only if no two stacks have the same starting address and $(x_{n+1}-x_1) >> n$, where n is the number of stacks.

Proof of Proposition 1A: Consider two cases.

(i) $S_s = \emptyset$. (All relocatable stacks must be relocated.)

Choose $\ell_1' = 0$, and choose $\ell_2', \ell_3', \ldots, \ell_n'$ so that stacks $2,3,\ldots,n$ must all be moved down.*

(ii) $S_s \neq \emptyset$. (Some stacks can remain stationary.)

$\forall_i \in S_s \cup \{1\}$ choose $\ell_i' = 0$, and then $\forall_j \in (N_r - S_r)$ choose ℓ_j' large enough so that all such stacks j have to be moved down.*

Since, in both cases, S_s is the only solution (by construction) it is the max-cost solution.

*Note that in the case where ℓ_1 already is zero and $x_2 = x_1+1$, then the solution $S_s = \emptyset$ does not preserve condition X since after relocation $x_1 = x_2$.

1.2 Defining the Solution Lattice

Definitions:[*] A relation $\mathscr{S}$ in a set A is a strong order relation if and only if it is irreflexive and transitive; a relation $\mathscr{W}$ in A is a weak order relation if and only if it is reflexive, anti-symmetric, and transitive. An ordered set is a triple $\langle A, \mathscr{S}, \mathscr{W} \rangle$ consisting of a set, and corresponding strong and weak order relations. A weak order relation $\mathscr{W}$ in A is linear if and only if $(\forall\, a,b \in A)$ $a\mathscr{W}b$ or $b\mathscr{W}a$, a strong order relation is linear if and only if $(\forall\, a,b \in A)$ $a\mathscr{S}b$, $a = b$, or $b\mathscr{S}a$.

Definition: (E,L,&P) For a given memory relocation problem $M = (X_r, L')$, define a binary relation R on the set $N' = N_r \cup \{1, n+1\}$ as follows: for $i,j \in N'$, $x_i, x_j \in X'$, where $X' = X_r \cup \{1, n+1\}$, $\ell'_k \in L'$, and $i < j$,

$$(i,j) \in R \iff \left\{ \left[\sum_{k=i}^{j-1} \ell'_k \right] \leq (x_j - x_i) \right\}.$$

For $i > j$,

$$(i,j) \in R \iff \left\{ \left[\sum_{k=j}^{i-1} \ell'_k \right] \leq (x_i - x_j) \right\}.$$

The relation R will occasionally be written as $R(M)$, or $R(X_r, L')$ to emphasize that the relation encodes a specific relocation problem.

*The algebraic definitions used in this section are from (G) Chapter 6.

<u>Claim</u>: The relation R is irreflexive, symmetric, and transitive.

<u>Claim</u>: Given M, $\vec{R}$ and $\vec{R}_r$ are corresponding strong and weak order relations where $\vec{R} = \{(i,j) \mid i < j$ and $(i,j) \in R(M)\}$ and $\vec{R}_r = \vec{R} + \{(i,i) \mid i \in N'\}$.

<u>Claim</u>: Given M, $<N',\vec{R},\vec{R}_r>$ is an ordered set.

<u>Claim</u>. Given M, there is at least one subset of stacks $S' \subseteq N'$ such that $<S',\vec{R} \cap (S' \times S'),\vec{R}_r \cap (S' \times S')>$ is a linearly ordered set.

<u>Proof</u>. By constraint (3), (Capacity), $S' = \{1,n+1\}$ is such a subset.*

<u>Proposition 2</u>: Given M, let $S \subseteq N'$ be any subset of stacks such that $|S| \geq 2$ and $<S,\vec{R} \cap (S \times S),\vec{R}_r \cap (S \times S)$ is linearly ordered. For all pairs of stacks $s_1,s_2 \in S$, such that $s_1 < s_2$, s_1 and s_2 need not be relocated relative to each other, that is, there is room for the memory requirements of the s_2-s_1 stacks $s_1,s_1+1,s_1+2,\ldots,s_2-1$ between s_1 and s_2-1 inclusive.

<u>Proof</u>. If S is a linearly ordered set, then $R(M)$ is satisfied for all pairs of stacks in S. Thus no relocation is necessary for all pairs in S <u>relative to each other</u>.

*If $S' = \{1,n+1\}$ is the only such linearly ordered set, for a given M, then the solution (and max-cost solution) $S_s \subseteq N_r$ to the problem (w,M) is $S_s = \emptyset$, the empty set.

Comment on Proposition 2:

1) If $|S| > 2$, then, in general, some of the stacks between s_1 and s_2, exclusive, will not be elements of S,
2) in which case, they must be relocated if s_1 and s_2 are to remain in place,
3) and the fact that any such stack lies between two linearly ordered stacks, s_1 and s_2, implies that there is sufficient space for a successful relocation of all stacks which are between s_1 and s_2, exclusive, and not in S, while s_1 and s_2 remain stationary.

Proposition 3: Let $S_s' = S_s \cup \{1,n+1\}$ where $S_s \subseteq N_r$ is any subset of relocatable stacks. Given a memory relocation problem M, $\langle S_s', \vec{R} \cap (S_s' \times S_s'), \vec{R}_r \cap (S_s' \times S_s') \rangle$ is a maximal linearly ordered set if and only if S_s is a solution to M.

Proof. ($\Longrightarrow$) If S_s' is linearly ordered, then by Proposition 2 no element of S_s' needs to be relocated, since $1, n+1 \in S_s'$. If S_s' is maximal, then no properly containing set can remain stationary. Therefore, S_s is a solution.

($\Longleftarrow$) If S_s is a solution, then all pairs in S_s' satisfy $R(M)$, so that S_s' is linearly ordered. Thus, S_s' is a maximal set with this property by definition of "solution."

Definition. Let $\mathcal{W}$ be a weak order relation in A and let $B \subseteq A$. The element $c \in A$ is an upper bound of B if $(\forall b \in B)\ b\mathcal{W}c$. Similarly, d is a lower bound of B if $(\forall b \in B)\ d\mathcal{W}b$. If there exists at most one upper bound c_0 of B such that for all upper bounds c of B, $c_0\mathcal{W}c$, then c_0 is the supremum of B, or sup B. Similarly, if there exists at most one lower bound d_0 of B such that for all lower bounds d of B, $d\mathcal{W}d_0$, then d is the infinum of B, or inf B.

Definition. An ordered set $\langle A,\mathcal{S},\mathcal{W}\rangle$ is a lattice if and only if each two-element subset of A has both a sup and inf.

Definition. Given a memory relocation problem M, an augmented solution is a solution to M plus the stacks 1 and $n+1$. Thus, if S_s is a solution, $S_s' = S_s \cup \{1, n+1\}$ is an augmented solution.

Definition. Given M, let

$$N_\ell' = \{i \in N' \mid \exists S_s' \subseteq N' \text{ such that } i \in S_s'\}$$

where S_s' is any augmented solution to M, and $N' = N_r \cup \{1, n+1\}$.

Proposition 4: The triple

$$<N'_\ell, \vec{R} \cap (N'_\ell \times N'_\ell), \vec{R}_r \cap (N'_\ell \times N'_\ell)>$$

is a lattice.

Proof. Let $T = \{a,b\}$ be any two-element subset of N'_ℓ.* Assume $a < b$. Either a and b lie in the same augmented solution, or in different augmented solutions.

If they lie in the same augmented solution, then under the weak order relation $\vec{R}_r$, $\sup\{a,b\} = b$ and $\inf\{a,b\} = a$.

If they lie in different augmented solutions, then $\{a,b\}$ always has at least one upper bound, the element $n+1$, and at least one lower bound, the element 1, under the relation $\vec{R}_r$. Thus there is always a unique least upper bound and a unique greatest lower bound, since $\vec{R}_r$ is antisymmetric. In other words, in the case where stacks a and b are in distinct augmented solutions, $\sup\{a,b\}$ is the least stack above a and b common to both augmented solutions, and $\inf\{a,b\}$ is the greatest stack below a and b common to both augmented solutions.

*Again by the capacity constraint, for all M, $|N'| \geq 2$.

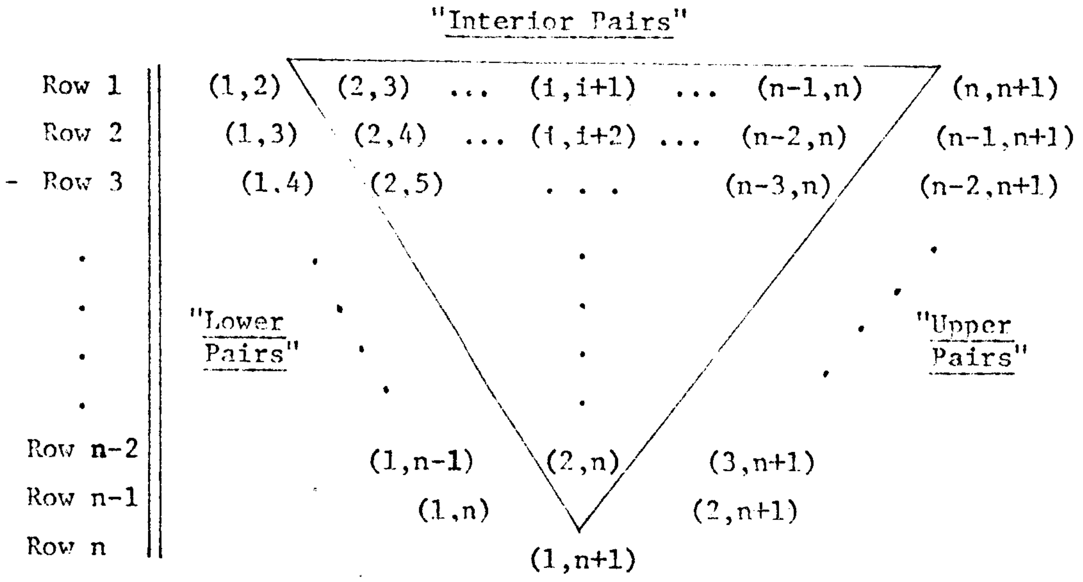
Definition. Let the lattice defined by the set $N' = \{1,2,\ldots,n+1\}$ and the relation $\vec{R}_r(M)$ be called the solution lattice.

Claim. Given $\vec{R}_r(M)$ and constraints (1)-(4), there is one and only one solution lattice.*

*Note that this assertion of uniqueness requires the fact that the elements of the lattice are labeled with integers.

1.3 Constructing the Solution Lattice

Let the non-identical pairs of elements of the set $N' = \{1,2,\ldots,n,n+1\}$ be arranged in rows as shown below.



Data Base for Solution Lattice Algorithm

In Row k, all pairs $(i,i+k)$, for $1 \le i \le [(n+1)-k]$, appear arranged in ascending order of left element. Let each pair $\lceil(i,j)\rceil$ represent the predicate, "given an M, the pair is an element of $\vec{R}_r$," i.e.,

$$\lceil(i,j)\rceil \Longleftrightarrow \lceil(i,j) \in \vec{R}_r(M)\rceil \ .$$

Proposition 5: Given M, for all $i \in N_r$, if either $\lceil(1,i)\rceil$ or $\lceil(i,n+1)\rceil$ are false, then stack i cannot be a member of a linearly ordered set containing stacks 1 and $n+1$; that is, it cannot be in a solution to M.

Proof. Assume $\lceil(1,i)\rceil$ is false and that $i \in S_s$, where S_s is a solution to M. Then $i \in S'_s = S_s \cup \{1,n+1\}$ which is linearly ordered, contradicting $(1,i) \notin \vec{R}_r(M)$. Similarly, if $\lceil(i,n+1)\rceil$ is false, and i is a member of a solution S_s, then $i \in S'_s = S_s \cup \{1,n+1\}$ which is linearly ordered, contradicting $(i,n+1) \notin \vec{R}_r(M)$.

Algorithm for Constructing the Solution Lattice

The algorithm constructs the solution lattice proceeding row by row coloring pairs according to the following scheme.

Green--A pair (i,j) is colored green if $(i,j) \in \vec{R}_r(M)$, and there are no pairs $(i,k),(k,j) \in \vec{R}_r$.

Yellow--A pair (i,j) is colored yellow if $(i,j) \in \vec{R}_r(M)$ by transitivity (namely, there are pairs $(i,k),(k,j) \in \vec{R}_r(M)$).

Red--A pair (i,j) is colored red if $(i,j) \notin \vec{R}_r(M)$.

Black--A "lower pair"* $(1,j)$ shall be colored black if $(j,n+1) \notin \vec{R}(M)$, i.e., $(j,n+1)$ is colored red. An "upper pair"* $(j,n+1)$ shall be colored black if $(1,j) \notin \vec{R}(M)$, i.e., $(1,j)$ is colored red. An "interior pair"* (i,j) shall be colored black if any of $(1,i)$, $(1,j)$, $(i,n+1)$, $(j,n+1)$ fail to be true, i.e., if any of them are colored red.

Rule: If a pair is colored more than once, the most recent coloring takes precedence.

Initialization: Color the pair $(1,n+1)$ green since constraint (3) specifies that memory itself cannot overflow.

Solution Lattice Algorithm:

```
INITIALIZE:    $k \leftarrow 0$

INCREMENT:     $k \leftarrow k+1$        ... k is Row index

LOWER.PAIR:    If $(1,1+k)$ is already black, go to UPPER.
     PAIR. Otherwise, color $(1,1+k)$ green or red as
     appropriate. If $(1,1+k)$ is red, color $(1+k,n+1)$
     and all interior pairs $(i,1+k)$ and $(1+k,i)$ black
     regardless of row or prior color.

UPPER.PAIR:    If $(n+1-k,n+1)$ is already black go to
     INTERIOR.PAIRS. Otherwise, color $(n+1-k,n+1)$ green
     or red as appropriate. If $(n+1-k,n+1)$ is red,
     color $(1,n+1-k)$ and all interior pairs $(i,n+1-k)$
     and $(n+1-k,i)$ black regardless of row or prior color.
```

*See diagram at beginning of Section 1.3.

INTERIOR.PAIRS: Color the non-colored (i.e., non-black, non-yellow) pairs $(i,i+k)$, $i \in N_r$, green or red as appropriate.

TRANSITIVITY: In all rows k', where $k' > k$, the row index, color yellow all pairs which are true by transitivity.

TEST: If $(1,n+1)$ colored yellow then go to DONE. If $k = n-1$ then go to DONE. Otherwise, go to INCREMENT.

DONE: Stop.

Claim. Given that the solution lattice algorithm has terminated, let

$$N = \{i \in N' \mid (i,j) \text{ or } (j,i) \text{ colored green}\}.$$

Then $\langle N, \vec{R} \cap (N \times N), \vec{R}_r \cap (N \times N)\rangle$ is the solution lattice.

Proof: (of correctness of algorithm)

Follows from...

1) Proposition 5.

2) Definition of colors.

3) Order in which rows of pairs are colored.

1.4 Searching the Solution Lattice

Assume the solution lattice algorithm has been applied.

Definition. A sequence of pairs is a non-empty ordered set of pairs (i,j), where the left element of the first pair is 1 and the right element of the last pair is $n+1$ and every adjacent pair of pairs in the sequence is of the form $(i,j)(j,k)$.

Example: $SP = [(1,3),(3,4),(4,9),(9,10)]$, where $n = 9$.

Definition. $\{SP\} = \{i \in N' \mid (i,j)$ or $(j,i) \in SP\}$ where SP is a sequence of pairs.

Claim. Let SP be a sequence of pairs. $\{SP\}$ is a maximal linearly ordered set if and only if each pair in SP is colored green.

Proof. Given correctness of solution lattice algorithm, the claim follows from definition of sequence of pairs.

Definition. The smallest interval and largest interval are, respectively, the green pair (i,j) such that $j-i$ is minimum over all green pairs, and the green pair (k,ℓ) such that $\ell-k$ is maximum over all green pairs.

Claim. If (i,j) is a smallest interval and s is the index of the least row containing a green pair, then $s = j-i$. Similarly, $\ell = j-i$ if (i,j) is a largest interval and ℓ is the index of the greatest row containing a green pair.

At this point at least two alternatives present themselves. Either the solution lattice can be searched for a maximum cost solution, or the lattice constructor algorithm (Section 1.3) can be modified to search for a max-cost colution as it constructs the solution lattice. In the former (two-pass) case, the smallest and largest interval can be used to cut down on the search. Specifically, if the lattice constructor algorithm calculates the largest interval I_ℓ, then for a given green interior pair (i,j) one need only search for green pairs (j,k) where $j < k < j+I_\ell-1$. In the latter (one-pass) case, the modifications to the lattice constructor algorithm are as follows. In order to search for a maximum cost solution simultaneously the algorithm must associate with each interior pair (i,j) which is colored green an "up cost" and "down cost." These two costs are simple $w(\{SP_u\})_{max}$ and $w(\{SP_d\})_{max}$, respectively, where SP_u and SP_d are (possibly empty) partially formed sequences of pairs from the green pair (i,j) to, respectively, the green pairs $(k,n+1)$ and $(1,g)$. Thus, if at some point of the construction of the solution lattice there are one or more partial sequences of green pairs to the top or bottom of memory from a given green pair one can associated with that pair a cost which is maximum over the partial sequence of pairs. As partial sequences become sequences one need only keep the largest cost solution found up to that point.

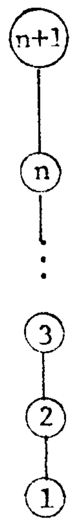
1.5 Time and Space Bounds

Definition. One unit of time is required by the lattice constructor algorithm to evaluate (color) one pair. (All other operations will be assumed to consume an amount of time which varies neither with respect to the complexity of a particular problem or the number of stacks in memory.)

Definition. One unit of memory is required to store one pair and all necessary information about that pair, such as its color.

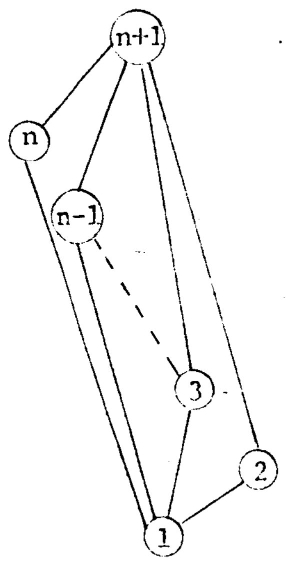
Assume for simplicity that the entire triangle of pairs is stored for manipulation by the lattice constructor algorithm. (Certainly no more than kn^2 "units of memory" will be required to store this array, where n is the number of stacks currently residing in memory.)

Consider two relocation optimization problems, $O_1 = (w,M_1)$ and $O_2 = (w,M_2)$. Let $M_1 = (X_r,L')$ be such that no relocation is required to accomodate the new memory requirements L'. Thus the solution S_s (which is also the only solution and thus the max-cost solution) is the set N_r --all the stacks can remain in place. The lattice constructor algorithm will require n time units to find this solution (the algorithm halts after examining one row). We can represent the solution as a graph on the set $N_r \cup \{1,r+1\}$. Adjacent stacks in the solution are connected by a single edge.



Solution to O_1

Let M_2 be such that there are $|N_r| = n-1$ solutions. The graph for this case is



Solutions to O_2

where each two-edge path from node ① to node (n+1) enumerates a solution.

In other words, any stack $i \in N_r$ can remain in place but the set $N_r - \{i\}$ has to be relocated. Note that no pair can be colored green by transitivity. (The pair $(1,n+1)$ is green initially.) Thus every pair in every row must be evaluated by the lattice constructor algorithm. This will consume $k(\frac{n(n-1)}{2})$ time units.

Claim. Given that the entire triangle of pairs is stored, O_1 and O_2 represent achievable lower and upper bounds on the "time" to construct the solution lattice.

Note that by information theory at least $\log_2 \mathcal{P}(N_r) = \log_2(2^{n-1}) = n-1$ binary decisions are required to find a (max-cost) solution, where $\mathcal{P}$ denotes the power set. Thus the lower bound on time comes within one time unit of achieving the theoretical minimum time, assuming one binary decision per time unit.

2.0 A Graph Theoretic Result

2.1 Defining a Permutation Graph

Definition. Let $P = [P(1), P(2), \ldots, P(n+1)]$ be a permutation of the positive integers $1, 2, \ldots, n+1$. Let $N' = \{1, 2, \ldots, n+1\}$ and Π be a subset of $N' \times N'$ defined as follows:

$$\Pi = \{(i,j) \mid [i < j \text{ and } P^{-1}(i) > P^{-1}(j)] \text{ or } [i > j \text{ and } P^{-1}(i) < P^{-1}(j)]\}$$

where $P^{-1}(i)$ is the element of N' which P maps into i.

Definition. A permutation graph $G(N', \Pi)$ is an undirected graph whose vertices are $1, 2, \ldots, n+1$ and whose edges are specified by the relation Π.

Theorem (E,L,&P): Given the relation R as defined in Section 1.2, $G(N', R)$ is a permutation graph.

2.2 Lattices Embedded in a Permutation Graph

Definition. Given a graph $G(N,R)$, two vertices $i,j \in N$ are adjacent if $(i,j) \in R$.

Definition. Let i and j be two adjacent vertices, such that $i \bar{<} j$, in a permutation graph $G(N',R)$. Let

$$G_\ell = \{k \mid k \text{ is a vertex on an ordered path* from } i \text{ to } j, \text{ inclusive, in } G(N',R)\}.$$

Theorem. If $\vec{R}$, $\vec{R}_r$ and G_ℓ are as defined previously,

$$<G_\ell, \vec{R} \cap (G_\ell \times G_\ell), \vec{R}_r \cap (G_\ell \times G_\ell)>$$

is a lattice.

Comment. The theorem above asserts that the ordered paths between two adjacent vertices in a permutation graph form a lattice.

Proof. An ordered path is a linearly ordered set under $\vec{R}$ and $\vec{R}_r$. Thus the theorem is simply a generalization of Proposition 4.

---

*Ordered paths $\Rightarrow$ a path on which the vertices are passed in ascending or descending order of their vertex labels.

Bibliography

(E,L,&P) Even, S. Lempel, A. and Pnueli, A., "Permutation Graphs and Transitive Graphs," Research Report SRRC-RR-70-11, Sperry Rand, Sudbury, Mass., February, 1970. J. ACM (to appear). Also in: Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems, 1970, Dept. of Elect. Engineering, Princeton University, Princeton, N.J., pp. 463-467.

(G) Gleason, A.M., Fundamentals of Abstract Analysis, Addison Wesley Publishing Co., Reading, Mass., 1966, Chapter 6.

(K) Knuth, D., The Art of Computer Programming/Fundamental Algorithms, Vol. 1, Addison Wesley, Reading, Mass., 1969, Chapter 2.2.2.